SMT-based Parameter Synthesis for L/U Automata

Michał Knapik[1,2] and Wojciech Penczek[1,3]

[1] Institute of Computer Science, Polish Academy of Sciences,
J.K. Ordona 21, 01-237 Warszawa, Poland
[2] International PhD Project in Intelligent Computing (MPD, FNP)
mknapik@ipipan.waw.pl
[3] Institute of Informatics, University of Natural Sciences and Humanities,
3 Maja 54, 08-110 Siedlce, Poland
penczek@ipipan.waw.pl

Abstract We present and evaluate a straightforward method of representing finite runs of a parametric timed automaton (PTA) by means of formulae accepted by satisfiability modulo theories (SMT)-solvers. Our method is applied to the problem of parametric reachability, i.e., the synthesis of a set of the parameter substitutions under which a state satisfying a given property is reachable. While the problem of parametric reachability is not decidable in general, we provide algorithms for under-approximation of the solution to this problem for a certain class of PTA, namely for the lower/upper bound automata.

1 Introduction

Model checking of real-time systems (RTS), performed by the analysis of their models is a very important subject of research. This is highly motivated by an increasing demand to verify safety critical systems, i.e., time-dependent systems, failure of which could cause dramatic consequences for both people and hardware. These systems include robotic surgery machines, nuclear reactor control systems, railway signalling, breaking systems, air traffic control systems, flight planning systems, rocket range launch safety systems, and many others. Humans already benefit a lot from a variety of real-time systems, being often unaware of this. Parametric model checking [2,10] aims at extending the successful developments of model checking of RTS. In this case, the model contains free variables, called parameters. Such a situation typically arises at the initial stages of a system design, when some of the details might be unknown.

Timed automata (TA) [1] constitute the most popular and applied class of RTS. The introduction of free variables into timed automata leads to parametric timed automata [2]. It was proven in [12] that the emptiness problem: "Is there a parameter valuation for which an automaton has an accepting run" for PTA is

W. Penczek is partly supported by the Polish National Science Centre under grant No. DEC-2011/01/B/ ST6/01477.

undecidable. Several tools have been implemented ([3,5,9,12,14,16]), which allow to verify certain properties of PTA (or related phase automata in [9]). All these tools except for [9] have one thing in common: they aim at fully describing the set of parameter substitutions under which the given property holds. Unfortunately, this means that the process of parametric model checking does not need to stop, consuming time and memory resources. These approaches usually employ extensions of classical (non-parametric) model checking methods such as: parametric difference bound matrices [5,12], partition refinement [16], compositional model-checking [9], and CEGAR and CEGAR-inspired methods [3,4,11].

In [17] a theoretical basis for bounded model checking for PTA was introduced. We have presented the counterpart of the region graph, which allows for a synthesis of a part of the set of constraints under which the given existential CTL property holds. The proposed method ensures that the process of verification stops with correct (but usually not complete) results. In the current paper we continue this work in order to ensure its feasibility. To this aim we consider lower bound/upper bound (L/U) automata [12] in which each parameter occurs either as a lower- or as an upper bound in the timing constraints. Despite this limitation, L/U automata are still interesting in practice, as for example they can be used to model the Fischer's Mutual Exclusion Algorithm, the Root Contention Protocol [12] and other well known systems[1]. Hune et al. [12] showed that the emptiness problem for L/U automata with respect to finite runs is decidable, whereas Bozelli and Torre [8] proved that it is also decidable w.r.t. infinite accepting runs. Similarly, the universality problem: "Does an automaton have an accepting run for each parameter valuation" is decidable for L/U automata. The above decidability results do not solve the problem of finding the valuations of the parameters in case the answer to the emptiness problem is positive and the answer to the universality problem is negative. This means that in case an automaton does have an accepting run only for some parameter valuations, it is not known how to compute them. The above observation is the main motivation of our work, which aims at offering a symbolic method for the synthesis of parameter valuations for which an L/U automaton satisfies some reachability property, i.e., it has a finite accepting run.

From the practical point of view the synthesis of all the parameter valuations is usually not needed: an analyst would typically be satisfied with a possibility of obtaining just a part of them. The direct analysis of parametric region graph is not feasible due to its typically large size, therefore the new methods of unwinding of the state space are needed. To this end, in this paper we offer a direct translation from an unwinding of the the concrete model of an L/U automaton to an SMT instance. This allows for synthesizing a subset of the parameter valuations for which an automaton satisfies some reachability property.

The rest of the paper is organized as follows. In the next section we briefly present the theory of parametric timed automata and formulate the task of parametric synthesis. In Section 3 we present how to encode all the finite runs of a

[1] IEEE Computer Society. IEEE Standard for a High Performance Serial Bus. Std 1394-1995, 1996.

given length as an SMT formula, and in Section 4 the algorithm for a state space exploration and a parameter synthesis for L/U automata is presented. Section 5 contains the preliminary evaluation of our method, as applied to two benchmarks: Fischer's Mutual Exclusion Protocol and a version of Generic Timed Pipeline Paradigm. We conclude with a brief discussion in Section 5.

2 Theory of parametric timed automata

In this section we introduce all the main notions and define timed automata, parametric timed automata, and L/U automata.

Parametric timed automata, to be defined later, employ two sets of variables: the set $X = \{x_1, \ldots, x_n\}$ of real time variables, called *clocks*, and the set $P = \{p_1, \ldots, p_m\}$ of integer variables, called *parameters*. Both types of the variables are used in the clock constraints of parametric timed automata.

The clock constraints are built using *linear expressions*, i.e., expressions of the form $\sum_{i=1}^{m} t_i \cdot p_i + t_0$, where $t_i \in \mathbb{Z}$. Clocks or differences of clocks compared with linear expressions, formally, the expressions of the form $x_i \prec e$ or $x_i - x_j \prec e$, where $i \neq j$, $\prec \in \{\leq, <\}$ and e is a linear expression, are called *simple guards*. The conjunctions of simple guards are called *guards*. By G we denote the set of all guards. By G' we mean the subset of G consisting of the guards built only of the simple guards of type $x_i \prec e$, where $x_i \in X$ and $\prec \in \{\leq, <\}$.

The clocks range over the nonnegative reals ($\mathbb{R}_{\geq 0}$) while the parameters range over the naturals ($\mathbb{N}$, including 0). The function $\upsilon : P \to \mathbb{N}$ is called a *parameter valuation* and $\omega : X \to \mathbb{R}_{\geq 0}$ is called a *clock valuation*. Sometimes it is convenient to perceive υ and ω as points in, respectively, $\mathbb{N}^m$ and $\mathbb{R}^n_{\geq 0}$.

The value obtained by substituting the parameters in a linear expression e according to the parameter valuation υ is denoted by $e[\upsilon]$. If $\omega(x_i) - \omega(x_j) \prec e[\upsilon]$ ($\omega(x_i) \prec e[\upsilon]$) holds, then we write $\omega \models_\upsilon x_i - x_j \prec e$ (resp., $\omega \models_\upsilon x_i \prec e$). This notion is naturally extended to the guards.

Two operations can be executed on the clocks: incrementation and reset. Let ω be a clock valuation and $\delta \in \mathbb{R}$, then by $\omega + \delta$ we denote such a clock valuation that $(\omega + \delta)(x_i) = \omega(x_i) + \delta$ for all $1 \leq i \leq n$. A set of the expressions of the form $x_i := b_i$, where $b_i \in \mathbb{N}$, and $1 \leq i \leq n$ is called a *reset*, and the set of all resets is denoted by R. Let ω be a clock valuation and r be a reset, then by $\omega[r]$ we denote such a clock valuation that $\omega[r](x_i) = b_i$ if $x_i := b_i \in r$, and $\omega[r](x_i) = \omega(x_i)$ otherwise. Intuitively, resetting a clock valuation amounts to setting the selected clocks to some fixed values, while leaving the remaining clocks intact. The *initial clock valuation* ω_0 satisfies $\omega_0(x_i) = 0$ for all $x_i \in X$.

2.1 Parametric timed automata

Timed automata [1] are an established formalism for modelling the behavior of real-time systems. The clock constraints are expressed as the restrictions imposed on clocks or differences of clocks. Parametric timed automata [2] are an extension of timed automata, where linear expressions containing parameters are allowed in the clock constraints.

Definition 1. *A* parametric timed automaton *is a seven-tuple* $\mathcal{A} = \langle Q, l_0, A, X, P, \rightarrow, I \rangle$*, where:*

- Q *is a finite set of* locations*,*
- $l_0 \in Q$ *is the* initial location*,*
- A *is a finite set of* actions*,*
- X *and* P *are, respectively, finite sets of clocks and parameters,*
- $I : Q \rightarrow G'$ *is an* invariant *function,*
- $\rightarrow \subseteq Q \times A \times G \times R \times Q$ *is a transition relation.*

A transition $(q, a, g, r, q') \in \rightarrow$ *is typicaly denoted by* $q \overset{a,g,r}{\rightarrow} q'$*.*

Notice that the co-domain of the invariant function is the conjunction of upper bounds on clocks. This assumption is taken from [12] in order to ensure that the set of time delays under which the automaton can stay in a given location is connected and contains 0 (if nonempty) for each parameter valuation.

The *concrete semantics* of a parametric timed automaton under a parameter valuation v is defined in the form of a labelled transition system.

Definition 2 (Concrete semantics). *Let* $\mathcal{A} = \langle Q, l_0, A, X, P, \rightarrow, I \rangle$ *be a parametric timed automaton and* v *be a parameter valuation. The labelled transition system for* $\mathcal{A}$ *under* v *is defined as the tuple* $[\![\mathcal{A}]\!]_v = \langle S, s_0, \mathbb{R}_{\geq 0} \cup A, \overset{d}{\rightarrow} \rangle$*, where:*

- $S = \{(l, \omega) \mid l \in Q,$ *and* ω *is a clock valuation such that* $\omega \models_v I(l)\}$*,*
- $s_0 = (l_0, \omega_0)$ *(we assume that* $\omega_0 \models_v I(l_0)$*),*
- *Let* $(l, \omega), (l', \omega') \in S$*. The transition relation* $\overset{d}{\rightarrow} \subseteq S \times S$ *is defined as follows:*
  - *if* $d \in \mathbb{R}_{\geq 0}$*, then* $(l, \omega) \overset{d}{\rightarrow} (l', \omega')$ *iff* $(l = l'$ *and* $\omega' = \omega + d)$*,*
  - *if* $d \in A$*, then* $(l, \omega) \overset{d}{\rightarrow} (l', \omega')$ *iff* $l \overset{d,g,r}{\rightarrow} l'$*, for some* $g \in G$*,* $r \in R$*, where* $\omega \models_v g$*, and* $\omega' = \omega[r]$*.*

The elements of S *are called the* concrete states *of* $[\![\mathcal{A}]\!]_v$*.*

After substituting the parameters in $\mathcal{A}$ according to a parameter valuation v we obtain the timed automaton, denoted by $\mathcal{A}_v$. The concrete semantics of $\mathcal{A}_v$ is usually denoted by $[\![\mathcal{A}_v]\!]$ and it is straightforward [12] to observe that $[\![\mathcal{A}_v]\!] = [\![\mathcal{A}]\!]_v$.

For $k \in \mathbb{N}$ by a *k-run* ρ^k in $[\![\mathcal{A}]\!]_v$ we mean a sequence of states and transitions: $\rho^k = s_0 \overset{d_0}{\rightarrow} s'_0 \overset{act_1}{\rightarrow} s_1 \overset{d_1}{\rightarrow} s'_1 \overset{act_2}{\rightarrow} \ldots \overset{d_{k-1}}{\rightarrow} s'_{k-1} \overset{act_k}{\rightarrow} s_k \overset{d_k}{\rightarrow} s'_k$, where $d_i \in \mathbb{R}_{\geq 0}$ and $act_i \in A$ for all $0 \leq i \leq k$. By a *run* we mean any k-run for $k \in \mathbb{N}$. We say that k is the *length* of ρ^k and s'_k is k–reachable in $[\![\mathcal{A}]\!]_v$.

2.2 Parametric reachability and synthesis

The original definition of parametric timed automata [2] contains a distinguished subset of the locations, called *final locations*. A run is *accepted* under a given valuation of the parameters if it ends with a final state. The question of the

emptiness of a set of the parameter valuations under which there exists an accepting run was shown in [2] to be undecidable.

Following [12,17] we present the results in the setting typical for model checking, where we distinguish the model and the property to be verified.

Definition 3. *Let $\mathcal{A} = \langle Q, l_0, A, X, P, \rightarrow, I\rangle$ be a parametric timed automaton. The* state formulae *are defined by the following grammar:*

$$\phi = l \mid x_i \prec b \mid x_i - x_j \prec b \mid \phi \wedge \phi \mid \neg\phi,$$

where $l \in Q$, $x_i, x_j \in X$, $\prec \in \{\leq, <\}$ and $b \in \mathbb{N}$.
We also refer to a state formula as to a property*. Let υ be a parameter valuation, $(l, \omega) \in [\![\mathcal{A}]\!]_\upsilon$, and let ϕ, ψ be state formulae. We define the validity of a state formula in a global states, denoted $(l, \omega) \models \phi$, inductively as follows:*

- $(l, \omega) \models l'$ *iff* $l = l'$,
- $(l, \omega) \models x_i \prec b$ *iff* $\omega \models_\upsilon x_i \prec b$, *and* $(l, \omega) \models x_i - x_j \prec b$ *iff* $\omega \models_\upsilon x_i - x_j \prec b$,
- $(l, \omega) \models \phi \wedge \psi$ *iff* $(l, \omega) \models \phi$ *and* $(l, \omega) \models \psi$,
- $(l, \omega) \models \neg\phi$ *iff not* $(l, \omega) \models \phi$.

Let υ be a parameter valuation and ϕ be a state formula. Let $k \in \mathbb{N}$, $d_j \in \mathbb{R}_{\geq 0}$ and $act_j \in A$ for all $1 \leq j \leq k$. Let $\rho^k = s_0 \overset{d_0}{\rightarrow} s'_0 \overset{act_1}{\rightarrow} s_1 \overset{d_1}{\rightarrow} s'_1 \overset{act_2}{\rightarrow} \ldots \overset{d_{k-1}}{\rightarrow} s'_{k-1} \overset{act_k}{\rightarrow} s_k \overset{d_k}{\rightarrow} s'_k$ be a k-run in $[\![\mathcal{A}]\!]_\upsilon$. If for some $k \in \mathbb{N}$ there exists a run ρ^k in $[\![\mathcal{A}]\!]_\upsilon$ such that $s'_k \models \phi$, then we say that a state satisfying ϕ is reachable in $\mathcal{A}$ under υ and write $[\![\mathcal{A}]\!]_\upsilon \models EF\phi$. The EF modality originates from Computation Tree Logic (CTL), where $EF\phi$ stands for "there exists a path such that eventually ϕ holds".

The task of parametric reachability, otherwise called the parameter synthesis problem, is formulated as follows.

Let $\mathcal{A}$ be parametric timed automaton and let ϕ be a state formula.
Automatically describe the set $\Gamma(\mathcal{A}, \phi) = \{\upsilon \mid [\![A]\!]_\upsilon \models EF\phi\}$.

As mentioned earlier, there is no decision procedure for checking whether $\Gamma(\mathcal{A}, \phi)$ is empty or contains all the parameter valuations, therefore we can not expect to be able to fully solve the parameter synthesis problem, at least in the general case.

2.3 L/U automata

Hune et al. have identified in [12] an important class of parametric timed automata for which the problem of the emptiness of $\Gamma(\mathcal{A}, \phi)$ is decidable. These are the lower/upper bound automata (L/U automata, for short), where additional constraints on the parameters are used.

In what follows if f is a function and B a subset of its domain, then $f_{|B}$ stands for the restriction of f to B.

Definition 4. *A* lower/upper bound automaton *is a parametric timed automaton* $\mathcal{A} = \langle Q, l_0, A, X, P, \rightarrow, I \rangle$ *satisfying the following conditions*

- $P = L \cup U$, *where* $L = \{\lambda_1, \ldots, \lambda_l\}$, $U = \{\mu_1, \ldots, \mu_u\}$, *and* $L \cap U = \emptyset$.
- *Each linear expression in the guards or the invariants of* $\mathcal{A}$ *can be written in form* $\sum_{i=1}^{l} l_i \cdot \lambda_i + \sum_{j=1}^{u} u_j \cdot \mu_j + t_0$, *where* $l_i, u_j, t_0 \in \mathbb{Z}$ *and* $l_i \leq 0$, $u_j \geq 0$ *for all* $1 \leq i \leq l$ *and* $1 \leq j \leq u$.

The elements of L *are called the* lower parameters *while the elements of* U *are called the* upper parameters.

Intuitively, in an L/U automaton the clock constraints can be uniformly relaxed by decreasing the values assigned to the lower parameters and increasing the values assigned to the upper parameters.

Let $\mathcal{A}$ be an L/U automaton and υ be a parameter valuation. Define $\lambda = \upsilon_{|L}$ and $\mu = \upsilon_{|U}$. If υ' is also a valuation of the parameters, $\lambda' = \upsilon'_{|L}$, $\mu' = \upsilon'_{|U}$, and $\forall_{\lambda_i \in L} \lambda'(\lambda_i) \leq \lambda(\lambda_i)$ and $\forall_{\mu_j \in U} \mu(\mu_j) \leq \mu'(\mu_j)$, then we write $\upsilon \leq \upsilon'$.

When it is convenient to define υ in terms of of λ and μ, we write $\upsilon = (\lambda, \mu)$.

Proposition 1 ([12]). *Let* $\mathcal{A}$ *be an L/U automaton,* ϕ *a state formula, and* υ, υ' *be parameter valuations such that* $\upsilon \leq \upsilon'$. *Then,* $[\![\mathcal{A}]\!]_\upsilon \models EF\phi$ *implies* $[\![\mathcal{A}]\!]_{\upsilon'} \models EF\phi$.

Assume that $\mathcal{A}$ is an L/U automaton. From the above lemma it follows that the problem of the emptiness of $\Gamma(\mathcal{A}, \phi)$ can be reduced to the problem of reachability of a state satisfying ϕ in the automaton obtained from $\mathcal{A}$ by substituting lower parameters with 0 and removing each guard or invariant containing at least one upper parameter. The latter, in terms of [12], is equivalent to substituting ∞ for each upper parameter.

3 Translation to SMT

The idea of encoding of system's behavior using the translation to propositional formulae originates from [7]. The techniques for SAT-based verification of various extensions of timed automata have evolved in parallel with these based on difference bound matrices. Usually, it is possible to translate only a part of a model to a logical formula, hence this method is applied for bounded model checking: a technique especially suited for seeking for bugs and unwanted behaviors.

SMT-solvers extend the capabilities of SAT-solvers by allowing for formulae of the first order over several built-in theories as an input. In our considerations, we use SMT-solvers to obtain the satisfiability and example models (valuations of the parameters) for formulae expressed using boolean variables and operators together with real-valued variables, linear arithmetic operators and relations. As we have decided to make the translation as straightforward as possible, in this experimental phase we have chosen to use SMT-lib ver. 2.0 [6] compliant solvers and rich logics allowing for linear arithmetic over real sort (e.g. QF_LRA).

Let $\mathcal{A} = \langle Q, l_0, A, X, P, \to, I\rangle$ be a parametric timed automaton. In this section we show how to encode, for a given $k \in \mathbb{N}$, all the k-runs of $\mathcal{A}_v$ for all the parameter valuations v, as the formula acceptable by SMT solvers such as CVC3. We start with the description of the sorts (types), the variables, and the additional predicates used.

3.1 Sorts and Predicates

We encode the locations of $\mathcal{A}$ by means of enumerating them using propositional expressions. For each $i \in \mathbb{N}$ let $\mathcal{BV}^i = \{bv_1^i, bv_2^i, \ldots, bv_{\lceil log|Q|\rceil}^i\}$ be a set of propositional variables. Let $\mathcal{BE}^i$ denote the set of all the propositional formulae over $\mathcal{BV}^i$. It is straightforward to notice for each $i \in \mathbb{N}$ we can define the function $loc\_enc^i : Q \to \mathcal{BE}^i$ assigning to each of the locations from Q the conjunction of the literals (variables or their negations) from $\mathcal{BV}^i$ such that $loc\_enc^i(l) \wedge loc\_enc^j(l')$ is false iff $i \neq j$ or $l \neq l'$. Intuitively, for $l \in Q$ and $i \in \mathbb{N}$, the formula $loc\_enc^i(l)$ can be interpreted as an encoding of the location l at the i–th step ($i \leq k$) of the k-runs, using variables from $\mathcal{BV}^i$.

Recall that $X = \{x_1, x_2, \ldots, x_n\}$ is a set of the clocks. For each $i \in \mathbb{N}$ let $X^i = \{x_1^i, x_2^i, \ldots, x_n^i\}$ be a set of real variables, where $X^i \cap X^j = \emptyset$ for $i \neq j$, and similarly, let $T = \{t_0, t_1, \ldots\}$ be a set of real variables. As previously, the variables from X^i are used to encode the clock valuations in the i–th steps of the k-runs with the help of the variables from T, which record the time delays between the consecutive actions.

Recall that $P = \{p_1, p_2, \ldots, p_m\}$ is a set of the parameters ranging over $\mathbb{N}$. With a slight notational abuse we treat P as a set of the variables of real sort. In the current version, SMT-lib standard does not allow for typecasts between reals and integers, therefore we need to use the predicate is_int to ensure that the variables from P hold integer values only (e.g., $is\_int(7.0)$ evaluates to true, while $is\_int(4.3)$ is false).

Summarizing, when considering the k-runs, we declare $Vars^k = \bigcup_{i=1}^k X^i \cup T \cup P$ to be real variables and $Bvars^k = \bigcup_{i=1}^k \mathcal{BV}^i$ to be boolean variables. Additionally, we define the formula

$$TypeCut^k = \bigwedge_{v \in Vars^k} v \geq 0 \wedge \bigwedge_{p \in P} is\_int(p)$$

which ensures that all used variables range over the appropriate sets.

3.2 Encoding the Transitions

In what follows, if η is a formula containing the free variables $a_1, a_2, \ldots, a_n$, then by $\eta[a_1 \leftarrow a_1', a_2 \leftarrow a_2', \ldots a_n \leftarrow a_n']$ we denote the formula obtained by substituting a_1' for a_1, a_2' for a_2, etc. in η. Additionally, we assume that there are no two transitions having the same label in $\mathcal{A}$. This assumption is not essential

for the translation[2], and it is used only to simplify the presentation of the results and the associated proofs.

Let $tr = l \overset{act,g,r}{\rightarrow} l'$ be a transition, where l, l' are respectively the source and the target location, act is the action label, g is the guard, and r is the reset. It is convenient to use the following notations: $source(tr) = l$, $target(tr) = l'$, $guard(tr) = g$, and $reset(tr) = r$. Now, let $i \in \mathbb{N}$. We define $guard^i(tr) = guard(tr)[x_1 \leftarrow x_1^i, \ldots, x_n \leftarrow x_n^i]$, i.e., the encoding of $guard(tr)$ using the variables introduced earlier. Define $reset^i(tr)$ as the smallest set such that $x_j^i := a + t_i \in reset^i(tr)$ if $x_j := a \in reset(tr)$ and $x_j^i := x_j^{i-1} + t_i \in reset^i(tr)$ otherwise, for each $1 \leq j \leq n$. Intuitively, $reset^i(tr)$ models the new value of each clock after the consecutive reset and delay. Let $s \in Q$ be a location, we define $inv^i(s) = I(s)[x_1 \leftarrow x_1^i, \ldots, x_n \leftarrow x_n^i]$, i.e., the encoding of the invariant of s. The above notions are combined to define the encoding of the transition tr as follows:

$$\begin{aligned} tr\_enc^i(tr) = {} & loc\_enc^i(source(tr)) \wedge guard^i(tr) \wedge reset^{i+1}(tr) \\ & \wedge inv^{i+1}(target(tr)) \wedge loc\_enc^{i+1}(target(tr)). \end{aligned}$$

The correctness of the above construction is stated in the following lemma.

Lemma 1. *Let $tr = l \overset{act,g,r}{\rightarrow} l'$ be a transition in $\mathcal{A}$, υ be a parameter valuation, (l, ω) be a concrete state in $[\![\mathcal{A}]\!]_\upsilon$ and $i \in \mathbb{N}$. Then, $(l, \omega) \xrightarrow{act} (l', \omega[r]) \xrightarrow{d} (l', \omega[r] + d)$ in $[\![\mathcal{A}]\!]_\upsilon$ iff for some valuation V of all the variables in $Vars^k$ satisfying $V \models tr\_enc^i(tr) \wedge TypeCut^{i+1}$ we have that:*

- $\upsilon = V_{|P}$,
- $\omega = V_{|X^i}[x_1^i \leftarrow x_1, \ldots, x_n^i \leftarrow x_n]$,
- $d = V(t_{i+1})$,
- $\omega[r] + d = V_{|X^{i+1}}[x_1^{i+1} \leftarrow x_1, \ldots, x_n^{i+1} \leftarrow x_n]$.

Proof. Observe that the locations are uniquely represented by their encodings, thus we can focus on nonboolean variables only.

($\Leftarrow$) Let V be a valuation of the variables such that $V \models tr\_enc^i(tr) \wedge TypeCut^{i+1}$. Let $\omega = V_{|X^i}[x_1^i \leftarrow x_1, \ldots, x_n^i \leftarrow x_n]$ and $\upsilon = V_{|P}$. Denote $\omega^i = V_{|X^i}$, then from $V \models guard^i(tr)$ we obtain $\omega^i \models_\upsilon guard^i(tr)$, which in turn yields that $\omega \models_\upsilon guard(tr)$. Let $d = V(t_{i+1})$, denote $\omega^{i+1} = V_{|X^{i+1}}$ and notice that from $V \models reset^{i+1}(tr)$ it follows that $\omega^{i+1}(x_j^{i+1}) = \omega^i[r](x_j^i) + d$ for all $1 \leq j \leq n$. Thus, if we denote $\omega' = V_{|X^{i+1}}[x_1^{i+1} \leftarrow x_1, \ldots, x_n^{i+1} \leftarrow x_n]$, then $\omega' = \omega[r] + d$. Now, observe that from $V \models inv^{i+1}(target(tr))$ we can infer that $\omega^{i+1} \models_\upsilon inv^{i+1}(target(tr))$, from which $\omega' \models_\upsilon I(target(tr))$, i.e., $\omega[r] + d \models_\upsilon I(target(tr))$. As $d \geq 0$ and in the view of the assumption that the invariants admit only upper bounds on clocks, we have also that $\omega[r] \models_\upsilon I(target(tr))$. This, together with the fact that $V \models TypeCut^{i+1}$ assures that the used variables range over the correct sets, concludes this part of the proof.

($\Rightarrow$) The implication in the other direction follows easily from the basic definitions of the transitions in $[\![\mathcal{A}_\upsilon]\!]$.

[2] We can always relabel the labels.

3.3 Encoding *k*-runs and reachability testing

Our aim is to encode all the k-runs of $\mathcal{A}_v$ for each parameter valuation v. Recall that n is the number of the clocks. To this end we define the following formula:

$$model\_enc^k(\mathcal{A}) = TypeCut^k \wedge \big(\bigwedge_{i=1}^{n}(x_i^0 = t_0) \wedge loc\_enc^0(l_0) \wedge inv^0(l_0)\big) \\ \wedge \bigwedge_{i=0}^{k-1} \bigvee_{tr \in \rightarrow} tr\_enc^i(tr).$$

The first component ensures that all variables range over the proper values. The second component sets all the initial clocks to some arbitrary common value (the assumption that the invariants represent the upper bounds on the clocks is significant here), encodes the initial state, and makes sure that its invariant is satisfied. The last component encodes all the possible transitions in the k–runs.

Let ϕ be a state formula and $i \in \mathbb{N}$. Let $\{l_1, \ldots, l_m\}$ be a set of all the locations present in ϕ. We define the encoding of ϕ as follows:

$$pr\_enc^i(\phi) = \phi[x_1 \leftarrow x_1^i, \ldots, x_n \leftarrow x_n^i, l_1 \leftarrow loc\_enc^i(l_1), \ldots, l_m \leftarrow loc\_enc^i(l_m)],$$

i.e., we simply substitute in ϕ each clock with its i–th variable counterpart, and each location with its encoding using boolean variables from $\mathcal{BV}^i$.

We obtain the formula to be used for testing and parameter synthesis by combining the encodings of the k-runs and the property, as presented in the following lemma.

Lemma 2. *Let $\mathcal{A}$ be a parametric timed automaton, ϕ be a state formula, v be a valuation of the parameters, and $k \in \mathbb{N}$. A state satisfying ϕ is k–reachable in $[\![\mathcal{A}]\!]_v$ iff there exists a valuation V of all the variables in $Vars^k$ such that $V \models model\_enc^k(\mathcal{A}) \wedge pr\_enc^k(\phi)$ and $V_{|P} = v$.*

Proof. Due to the presence of $TypeCut^k$ in $model\_enc^k(\mathcal{A})$ we know that all the variables range over the proper sets.

Let l be the (unique) location such that $V_{|\mathcal{BV}^k} \models loc\_enc^k(l)$ and $\omega^k = V_{|X^k}[x_1^k \leftarrow x_1, \ldots, x_n^k \leftarrow x_n]$, First, we prove that $V \models model\_enc^k(\mathcal{A})$ iff the state (l, ω^k) is k–reachable in $[\![\mathcal{A}_v]\!]$ for $v = V_{|P}$.

If $k = 0$, then $\bigwedge_{i=1}^{n}(x_i^0 = t_0) \wedge loc\_enc^0(l_0) \wedge inv^0(l_0)$ is satisfied by the valuation V iff V is such that if we denote $\omega^0 = V_{|X^0}[x_1^0 \leftarrow x_1, \ldots, x_n^0 \leftarrow x_n]$ and $V_{|P} = v$, then for some $t^0 = V(t_0)$ we have that $\omega^0 = \omega_0 + t^0$ and $\omega^0 \models_v I(l_0)$. This corresponds to the set of states to which $[\![\mathcal{A}_v]\!]$ can progress by the time transitions

For the inductive step observe that $model\_enc^k(\mathcal{A}) = model\_enc^{k-1}(\mathcal{A}) \wedge \bigvee_{tr \in \rightarrow} tr\_enc^{k-1}(tr) \wedge TypeCut^k = \bigvee_{tr \in \rightarrow} \big(model\_enc^{k-1}(\mathcal{A}) \wedge tr\_enc^{k-1}(tr)\big) \wedge TypeCut^k$ and apply Lemma 1 and the inductive assumption.

To conclude, notice that $pr\_enc^k(\phi)$ simply encodes all the concrete states for which ϕ holds, using the variables from $\mathcal{BV}^k \cup X^k$.

4 Parameter set approximations

We already know how to write, for a given parametric timed automaton and a property, the formula encoding k–reachable states for which the property holds together with the associated valuations of the parameters. It might be beneficial to verify this formula as it is, if we wish to obtain the answer to the question whether the property is satisfied by k–reachable states. We can also rely on the SMT-solver to obtain an exemplary witness, i.e., a correct valuation of the parameters. Our task is, however, to systematically explore the space of the admissible parameters, with a hope for painting a part of the picture from which an analyst can make further generalizations.

Let $\mathcal{A} = \langle Q, l_0, A, X, P, \to, I \rangle$ be an L/U automaton and $P = L \cup U$, where L and U are disjoint sets of the upper and the lower parameters, respectively. Assume that $L = \{\lambda_1, \ldots, \lambda_l\}$ and $U = \{\mu_1, \ldots, \mu_u\}$.

Let ϕ be a property and υ be such a valuation of the parameters that there exists a reachable state in $[\![\mathcal{A}]\!]_\upsilon$ satisfying ϕ. Recall (Proposition 1) that in the class of the L/U automata this means that a state satisfying ϕ is also reachable in $[\![\mathcal{A}]\!]_{\upsilon'}$ for each υ' such that $\upsilon \leq \upsilon'$.

Define the *complementing clause* with respect to υ as follows

$$ComplClause(\upsilon) = \bigvee_{i=1}^{l} (\lambda_i > \upsilon(\lambda_i)) \vee \bigvee_{i=1}^{u} (\mu_i < \upsilon(\mu_i)),$$

and observe that $\upsilon' \models ComplClause(\upsilon)$ iff $\upsilon \leq \upsilon'$ is not true.

We employ $ComplClause(\upsilon)$ to block the SMT-solver from seeking for parameter valuations which can be inferred from the L/U automata properties and the set of the parameters that has been already synthesized.

The following algorithm attemps to synthesise parameter valuations for which there exists a k–reachable state satisfying the property ϕ. If the search is successful, the user is presented with a newly synthesised parameter valuation υ and asked whether the procedure should be continued. If so, a new blocking $ComplClause(\upsilon)$ is added to the main formula and the loop takes another turn.

Algorithm 1 ReachSynth($\mathcal{A}, \phi, k$)

Input: an L/U automaton $\mathcal{A}$, a property ϕ, a depth value $k \in \mathbb{N}$
Output: a set *Res* of valuations of the parameters

1: $Res := \emptyset$
2: $reachFormula := model\_enc^k(\mathcal{A}) \wedge pr\_enc^k(\phi)$
3: **while** user requests to expand *Res* and reachFormula is satisfiable **do**
4: let V be such that $V \models reachFormula$ and $\upsilon := V_{|P}$
5: let $Res := Res \cup \{\upsilon\}$
6: let $reachFormula := reachFormula \wedge ComplClause(\upsilon)$
7: **end while**
8: **return** *Res*

Note that in the above algorithm the testing for satisfiability (line 3), and extraction of the witness valuation v of the parameters (line 4) is performed by means of a call to an external SMT-solver.

Lemma 3. *Let $\mathcal{A}$ be an L/U automaton, ϕ be a property, and $k \in \mathbb{N}$. For each valuation of the parameters v' such that there exists $v \in ReachSynth(\mathcal{A}, \phi, k)$ satisfying $v \leq v'$ we have that $[\![\mathcal{A}]\!]_{v'} \models EF\phi$.*

Proof. It follows immediately from Lemma 2 combined with the properties of $ComplClause(v)$.

The *ReachSynth* algorithm can be used as the main building block of a bounded parametric model checking process. The input consists of an L/U automaton $\mathcal{A}$ and a property ϕ. Initially, we can employ the results from [12] to solve the *emptiness problem* for ϕ and $\mathcal{A}$, i.e., to check whether there exists a parameter valuation v such that $[\![\mathcal{A}]\!]_v \models EF\phi$. If the existence of such a valuation is confirmed, then the *universality problem*, i.e., the question whether $[\![\mathcal{A}]\!]_v \models EF\phi$ for all parameter valuations v, can be checked as a dual to the emptiness. If the answer to the universality problem is false, then $ReachSynth(\mathcal{A}, \phi, k)$ is called, starting from $k = 0$ and incrementing the value of k whenever the previous call returned empty set or the loop was stopped by the user.

5 Evaluation

In this section we present some preliminary results on parametric analysis of two selected models, namely Fischer's Mutual Exclusion Protocol and a version of Generic Timed Pipeline Paradigm [15]. Both of them are well established and scalable benchmarks specified in a form of networks of parametric timed automata.

Definition 5. *Let $\mathcal{U} = \{\mathcal{A}^i = \langle Q^i, {l_0}^i, A^i, X^i, P^i, \rightarrow^i, I^i\rangle \mid 1 \leq i \leq m\}$ be a set (a* network*) of parametric timed automata such that $X^i \cap X^j = \emptyset$ for each $1 \leq i, j \leq m$ and $i \neq j$. Let $\mathcal{L}(a) = \{1 \leq i \leq m \mid a \in A^i\}$ be a function associating with each action $a \in \bigcup_{1\leq i\leq m} A^i$ the indices of the automata recognizing a. We define the* product automaton *$\mathcal{A} = \langle Q, l_0, A, X, P, \rightarrow, I\rangle$ of the network $\mathcal{U}$, where:*

- $Q = \prod_{i=1}^{m} Q^i$,
- $l_0 = (l_0^1, \ldots, l_0^m)$,
- $A = \bigcup_{i=1}^{m} A^i$,
- $X = \bigcup_{i=1}^{m} X^i$,
- $P = \bigcup_{i=1}^{m} P^i$,
- $I((l_1, \ldots, l_m)) = \bigwedge_{i=1}^{m} I^i(l_i)$ *for each* $(l_1, \ldots, l_m) \in Q$,

and the transition relation $\rightarrow$ is such that:

- $(l_1, \ldots, l_m) \overset{a,g,r}{\rightarrow} (l'_1, \ldots, l'_m)$ *iff for each $i \in \mathcal{L}(a)$ we have $l_i \overset{a,g_i,r_i}{\rightarrow_i} l'_i$, and $g = \bigwedge_{i\in\mathcal{L}(a)} g_i$, $r = \bigcup_{i\in\mathcal{L}(a)} r_i$, and $l_i = l'_i$ for all $i \in \{1, \ldots, m\} \setminus \mathcal{L}(a)$.*

In addition to a network, the user is expected to supply an *experiment's plan* file. Such a plan consists of a sequence of pairs (k, No) of natural numbers, where k is the length of the runs to be considered, and No is a maximal number of parameter valuations to be synthesised should the verified property be found satisfiable. Our tool goes through the pairs in accordance with increasing k, incrementally building the formulae to be tested as it was presented in earlier sections.

All the experiments have been performed on Intel P6200 2.13GHz dual core machine with 3GB memory, running Linux operating system.

5.1 Fischer's Mutual Exclusion Protocol

The timed automata network presented in Figure 1 models one of the possible solutions to the classical problem of mutual exclusion, i.e., ensuring that only one of the competing processes is able to gain an access to the critical section.

The system in question consists of n independent processes synchronised via the shared variable X. The model contains two parameters, i.e., the lower bound parameter δ and the upper bound parameter Δ. It is well known that no two processes are able to simultaneously get to their critical sections iff $\Delta \leq \delta$, thus we have chosen to verify the reachability of $\phi_1 = critical_1 \wedge critical_2$. Intuitively, this means that we aim to synthesise values of the parameters δ and Δ under which the system behaves incorrectly, allowing two competing processes to jointly enter their critical sections.

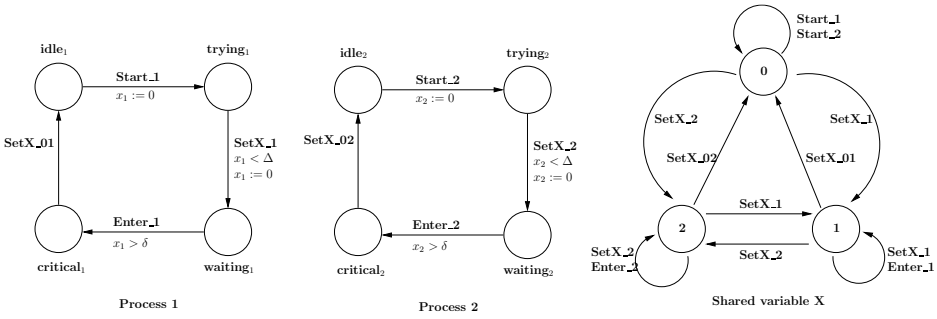

Figure 1: Fischer's Mutual Exclusion Protocol, 2 processes

As it turns out, for each test with the positive outcome, the set of the returned valuations consists of the pairs ($\delta = i$, $\Delta = i + 1$) for i from 0 to the limit No $= 10$ given in the experiment's plan. Clearly, from the point of view of an analyst and in light of Lemma 3 this result indeed justifies an educated guess that the mutual exclusion property is violated if $\Delta > \delta$.

n	k	SAT? (Y/N)	param vals found	max. form. size (MB)	form. bdg. time (sec.)	total CVC3 time (sec.)	peak CVC3 mem. (MB)
7	1–5	N	–	1.54	1.7	1.8	20
7	6	Y	10	3.51	2.25	37.2	41
8	1–5	N	–	2.95	3.95	3.6	30
8	6	Y	10	7.13	5.45	75.3	70
9	1–5	N	–	5.48	7.1	6.5	48
9	6	Y	10	13.71	11.86	187.8	119
10	1–5	N	–	9.43	13.1	11.33	69
10	6	Y	10	24.62	24.06	245.75	198
11	1–5	N	–	15.25	23.29	20.71	108
11	6	Y	10	41.84	46.23	504.35	331
12	1–5	N	–	24.94	40.13	25.11	170
12	6	Y	10	71.17	85.07	726.51	560
13	1–5	N	–	38.89	66.1	40.78	256
13	6	Y	10	115.8	149.24	1315.87	1000
14	1–5	N	–	57.76	105.98	67.50	384
14	6	Y	10	180.71	253.99	3192.47	1600

Table 1: Fischer's Mutual Exclusion parametric verification results

Legend: n: a number of competing processes, k: runs' lengths, *SAT?*: satisfiability, *4–th column*: the number of the synthesised parameter valuations, *5–th column*: the maximal (if k is an interval) size of the generated SMT-lib v2 formula, *6–th column*: the time spent on incrementally building the formulae, *7–th column*: the total time spent on verifying the formulae, *8–th column*: the maximal memory used by CVC3 solver.

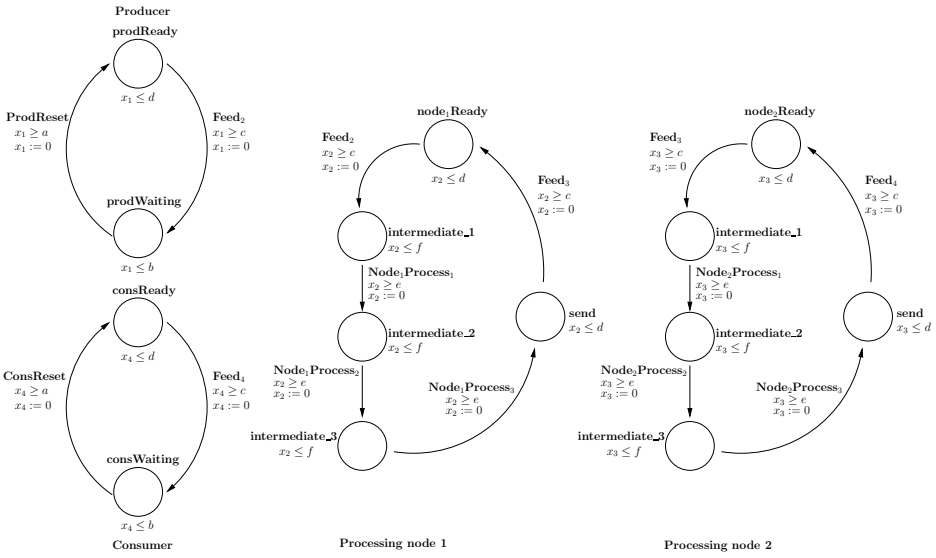

Figure 2: Generic Timed Pipeline Paradigm, 2 processing nodes of length 3

5.2 Generic Timed Pipeline Paradigm

The network presented in Figure 2 models the system consisting of the Producer feeding the Consumer with data sent through a sequence of nodes with additional processing capabilities.

The model is scalable with respect to the number n of the processing nodes and the length m of each processing node and it contains three lower (a, c, e) and three upper (b, d, f) parameters.

We have decided to add one dummy clock, called x_{total}, to the above system. It is straightforward to see that such an alteration does not change the behaviour of the model, and that x_{total} can be used to measure the total time passed along a given run. With the help of the new clock we have analysed the reachability of $\phi_2 = ConsWaiting \wedge ProdReady \wedge x_{total} \geq 5$. Again, the limit No is set to 10 for all k.

n	m	k	SAT? (Y/N)	param vals found	max. form. size (MB)	form. bdg. time (sec.)	total CVC3 time (sec.)	peak CVC3 mem. (MB)
2	10	1–12	N	—	0.01	0.01	1.27	7
2	10	13	Y	10	0.02	0.003	9.25	17
2	15	1–17	N	—	0.02	0.015	3.04	12
2	15	18	Y	10	0.02	0.004	22.29	29
2	20	1–22	N	—	0.03	0.02	6.40	18
2	20	23	Y	10	0.03	0.004	57.73	60
3	1	1–5	N	—	0.01	0.008	0.26	4
3	1	6	Y	10	0.02	0.004	16.91	26
3	2	1–7	N	—	0.02	0.014	0.51	5
3	2	8	Y	10	0.03	0.006	86.55	85
3	3	1–9	N	—	0.04	0.023	0.89	10
3	3	10	Y	10	0.05	0.008	13.68	17
3	4	1–11	N	—	0.06	0.034	1.48	10
3	4	12	Y	10	0.08	0.01	39.87	32
3	5	1–13	N	—	0.08	0.049	2.55	8
3	5	14	Y	10	0.11	0.014	2472.87	458

Table 2: Generic Timed Pipeline Paradigm parametric verification results
Legend: see Table 1, m: a number of processing nodes

Note that the generated SMT formulae are rather small in this case. This probably reflects the power of a concise representation by means of SMT instances rather than the size of model's state space [13].

Table 3 contains some exemplary parameter valuations, synthesised for several instances of the model. This illustrates the power of approximation-based approach, where the collected data may be used in search for general pattern.

n	m	k	a	b	c	d	e	f
2	15	18	0	0	0	0	0	0
			0	0	1	2	0	0
			1	1	0	1	0	1
			0	0	1	17	1	1
			0	0	2	16	0	0
			1	1	1	2	0	1
			0	0	2	19	1	1
			0	0	3	17	0	0
			1	1	0	15	1	1
			0	0	3	6	0	0
2	20	23	0	0	0	0	0	0
			0	0	1	2	0	0
			1	1	0	1	0	1
			0	0	1	22	1	1
			0	0	2	21	0	0
			1	1	0	20	1	1
			0	0	2	4	0	0
			1	1	1	2	0	1
			0	0	2	24	1	1
			0	0	3	22	0	0

n	m	k	a	b	c	d	e	f
3	1	6	0	0	0	1	0	0
			0	1	1	4	0	0
			1	1	0	1	0	0
			0	2	1	6	1	1
			1	1	1	4	0	0
			2	2	0	3	1	1
			0	1	0	2	1	1
			2	2	0	1	0	0
			1	1	0	3	1	1
			2	2	0	2	0	0
3	2	8	0	0	0	0	0	0
			0	1	1	3	0	0
			1	1	0	1	0	0
			0	3	1	7	1	1
			1	2	2	6	0	0
			2	2	0	1	0	1
			1	1	1	4	0	0
			1	1	1	3	0	0
			3	3	0	2	0	1
			0	2	0	4	1	1

n	m	k	a	b	c	d	e	f
3	3	10	0	0	0	0	0	0
			1	1	0	1	0	1
			2	2	0	2	0	2
			3	3	0	3	0	3
			4	4	0	4	0	4
			5	5	0	5	0	5
			6	6	0	6	0	6
			7	7	0	7	0	7
			8	8	0	8	0	8
			9	9	0	9	0	9
3	4	12	0	0	0	0	0	0
			1	1	0	1	0	1
			2	2	0	2	0	2
			3	3	0	3	0	3
			4	4	0	4	0	4
			5	5	0	5	0	5
			6	6	0	6	0	6
			7	7	0	7	0	7
			8	8	0	8	0	8
			9	9	0	9	0	9

Table 3: Generic Timed Pipeline Paradigm: exemplary parameter valuations

6 Conclusions

We have proposed a simple translation for a direct representation of finite runs of a parametric timed automaton in form of SMT instances. This translation coupled with blocking clauses allowed us for an underapproximation of the set of the parameter valuations under which the given reachability property holds in L/U automata. To the best of our knowledge this is the first such application of SMT solvers, and this is at the same time a proof-of-concept as well as a practical tool for exploring the spaces of parameter valuations.

In the future we plan to extend our work to the parametric verification of properties more complex than reachability, e.g., the existential fragment of CTL*. Additionally, we plan to investigate the possibilites of an automated inference of more general (or even complete) constraints under which the given property holds using partial knowledge on the space of the parameter valuations obtained using the methods presented in this paper.

References

1. R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.
2. R. Alur, T. Henzinger, and M. Vardi. Parametric real-time reasoning. In *Proc. of the 25th Ann. Symp. on Theory of Computing (STOC'93)*, pages 592–601. ACM, 1993.
3. É. André. Imitator ii: A tool for solving the good parameters problem in timed automata. In Yu-Fang Chen and Ahmed Rezine, editors, *INFINITY*, volume 39 of *EPTCS*, pages 91–99, 2010.
4. E. André, T. Chatain, E. Encrenaz, and L. Fribourg. An inverse method for parametric timed automata. *International Journal of Foundations of Computer Science*, 20(5):819–836, Oct 2009.
5. A. Annichini, A. Bouajjani, and M. Sighireanu. TREX: A tool for reachability analysis of complex systems. In *Proc. of the 13th International Conference on Computer Aided Verification*, CAV '01, pages 368–372, 2001.
6. C. Barrett, A. Stump, and C. Tinelli. The SMT-LIB Standard: Version 2.0. Technical report, Department of Computer Science, The University of Iowa, 2010. Available at `www.SMT-LIB.org`.
7. A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In *Proc. of the 5th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'99)*, volume 1579 of *LNCS*, pages 193–207. Springer-Verlag, 1999.
8. L. Bozzelli and S. La Torre. Decision problems for lower/upper bound parametric timed automata. *Formal Methods in System Design*, 35(2):121–151, 2009.
9. H. Dierks and J. Tapken. Moby/DC – A tool for model-checking parametric real-time specifications. In H. Garavel and J. Hatcliff, editors, *Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2003)*, volume 2619 of *LNCS*, pages 271–277. Springer, 2003.
10. L. Doyen. Robust parametric reachability for timed automata. *Inf. Process. Lett.*, 102:208–213, May 2007.

11. G. Frehse, S. K. Jha, and B. H. Krogh. A counterexample-guided approach to parameter synthesis for linear hybrid automata. In *Proc. of the 11th international workshop on Hybrid Systems: Computation and Control*, HSCC '08, pages 187–200, Berlin, Heidelberg, 2008. Springer-Verlag.
12. T. Hune, J. Romijn, M. Stoelinga, and F. Vaandrager. Linear parametric model checking of timed automata. *J. Log. Algebr. Program.*, 52-53:183–220, 2002.
13. M. Knapik, W. Penczek, M. Szreter, and A. Pólrola. Bounded parametric verification for distributed time Petri nets with discrete-time semantics. *Fundam. Inform.*, 101(1-2):9–27, 2010.
14. D. Lime, O. H. Roux, C. Seidner, and L. M. Traonouez. Romeo: A parametric model-checker for Petri nets with stopwatches. In S. Kowalewski and A. Philippou, editors, *TACAS*, volume 5505 of *Lecture Notes in Computer Science*, pages 54–57. Springer, 2009.
15. D. Peled. All from one, one for all: On model checking using representatives. In *Proc. of the 5th Int. Conf. on Computer Aided Verification (CAV'93)*, volume 697 of *LNCS*, pages 409–423. Springer-Verlag, 1993.
16. R. F. Lutje Spelberg and W. J. Toetenel. Splitting trees and partition refinement in real-time model checking. In *HICSS*, page 278, 2002.
17. W. Penczek and M. Knapik. Bounded Model Checking for Parametric Timed Automata. *T. Petri Nets and Other Models of Concurrency*, 5, to appear, 2012.